Short Tutorial: Backtracking With Recursion

Author **Message**

Shah-Cuber **Posted:** Thu Jun 25, 2009 5:56 pm Post subject: Short Tutorial: Backtracking With Recursion

Backtracking With Recursion

Obviously, out of summer boredom, there is nothing better than to make a tutorial on a random topic ... oh well, enjoy!

Several problems can be solved by combining recursion with backtracking. To begin understanding what it really is, think of a maze for example. When a dead end is reached we backtrack to the last place where there was a choice in paths to take, and take the other path (assuming there were only two possibilities). If this turns out also to be a dead end, we backtrack to the previous point where there was a choice.

This is a specific example of a general recursive approach to solving problems that involve searching through a space for an entity that satisfies a certain property. For traversing a maze, the space is the set of sequences of moves that can be made (drawn from turn left, turn right, move ahead without hitting a wall), and the property to satisfy is "start at the entry and arrive at the exit".

The requirements for a recursive solution are met in these ways.
.The problem is made smaller by taking one step in the space - by making one move.
.The base case occurs when the property has been fully satisfied and we arrive at the exit.
.The partial results are combined by putting the steps together in a way that is appropriate (perhaps printing them, or returning them as a list to some other program).

Backtracking is an essential problem solving tool for programs where there is not enough local information to know for certain how to progress toward the desired goal. Unlike, for example, merge sorting (or any other sorting algorithm), where it is known that sorting, and merging will make progress toward the desired result, a backtracking program for searching a solution space tries one way forward: it takes a legal step that extends the partial result obtained so far and does not invalidate the property that must be satisfied. It then makes a recursive call to look for a continuation of the partial result it has just extended.

Because the continuation may not exist, however, the search for it may fail. This occurs if the program tries a step forward that is taking it down a dead end street (the example of searching through a maze is a good one to keep in mind here). When a failure to find a continuation is reported, the program tries another legal step if it has one available to it, and looks for a continuation from that point. If all of the possible legal steps that it can try results in failed attempts to find a continuation, then the program must report a failure itself. This result is reported to the program that invoked it, which then, in turn, may have to try another legal step or report failure to its caller.

Now that this concept has been explained, I will introduce a problem that I have done between yesterday and today, that is a very good problem to explore for programmers (some of you may have already heard of this). So, another example of backtracking with recursion is the classic problem of placing 8 Queens on a chess board of 8x8 squares in such a way that no queen threatens another (http://en.wikipedia.org/wiki/Eight_queens_puzzle) (http://mathworld.wolfram.com/QueensProblem.html). This problem is not an inherently interesting part of computer science, but it does illustrate the general pattern for these recursive search and backtrack programs. The queen is the most versatile of all chess pieces. It can capture any piece in the same row, in the same column, or along either diagonal from its location on the board. These first two conditions mean that no two queens can have the same row number or the same column number. We must guard, as well, against threats from a diagonal.

```
Q . . . . . . . Q = Queen
. . . Q . . . .
. Q . . . . . .
. . . . Q . . .
. . Q . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

Queens are placed safely in column 1-5 but there is now no safe square in column 6 (sorry for the bad depiction >_>)

We begin our solution by placing the first queen in column 1, row 1. The second queen will be places in column 2. It obviously cannot be in row 1, but, because of a diagonal threat from the queen in column 1, it also cannot be row 2. So we place it in row 3 and continue trying to place the next queen in column 3. The smallest row there that is not threatened is row 5. Continuing to column 4, we can place a queen in row 2; in column 5 we can place a queen in row 4. But now in column 6 there is no square that is not threatened (see diagram). And we know that the solution, if there is one, must have a queen in each column.

So we must backtrack to column 5 and choose an alternative safe location there, if possible.

The only remaining safe location is row 8, so we try again to place a queen in row 6 with this change, and again we fail. So now we must backtrack to row 4 as row 5 has no more options. And so on...

To see how recursion can be used to solve the problem, we start by placing a queen in the first column and calling the recursive method to place a queen which, in turn, calls the same method to place the next queen, and so on until we reach the degenerate case where there is no columns in which a queen can be placed. We look for a recursive method that places a queen in a column assuming that queens have been placed successfully in the preceding columns. We do not know whether the most recent placement is a success until we try to place a queen in the next column. If we cannot, we must backtrack.

My solution to the Eight Queens problem uses a two dimensional Boolean array to represent the board where, if the value of a square is true it means there is a queen there and if the value of the square is false there is no queen there. I recommend exploring this concept further, and try solving the problem yourself (though there are numerous variations of this kind of problem), and submit anything you have discovered.
Here is the complete program in Java (I've been using python/ C++ too often nowadays). Hope you've enjoyed this short tutorial 🙂

code:

```
import java.util.Scanner;

/*
 * The Queens class.
 * Places a number of chess queens on a board of a
 * specified size so that no queen can attack another.
 */
    public class Queens
   {
      protected int boardSize;
      protected boolean[][] board;

       public Queens (int boardSize)
      {
         this.boardSize = boardSize;
         board = new boolean[boardSize][boardSize];

         for (int row = 0; row < boardSize; row++)
            for (int column = 0; column < boardSize; column++)
               board[row][column] = false;
      } // Queens constructor

       /*
        * pre: There are queens in columns 0 to (column - 1).
        * post: Queens have been placed in all the columns
        *       of board and PlaceQueens will return true, or
        *       PlaceQueens will return false.
```

```
  */

 protected boolean placeQueen (int column)
{
   int row;

   if (column == boardSize)
   {
      return true;
   }
   else
   {
      boolean successful = false;
      row = 0;
      while ((row < boardSize) && !successful)
      {
         if (threat (row, column))
         {
            row++;
         }
         else
         {
          // Place queen and try to place queen in next column.
            board[row][column] = true;
            successful = placeQueen (column + 1);

            if (!successful)
            {
               // Remove the queen placed in the column.
               board[row][column] = false;
               row++;
            }
         }
      }

      return successful;
   }
} // placeQueen method

 protected boolean threat (int row, int column)
{
  // Test for a queen on the same row.
   for (int c = 0; c < column; c++)
   {
      if (board[row][c])
      {
         return true;
      }
   }

  // Test for queen on up diagonal.
   for (int c = 1; c <= column; c++)
   {
      if (row < c)
      {
         break;
      }

      if (board[row - c][column - c])
      {
         return true;
      }
   }

  // Test for queen on down diagonal
   for (int c = 1; c <= column; c++)
   {
      if (row + c >= boardSize)
      {
         break;
      }

      if (board[row + c][column - c])
      {
         return true;
      }
   }

   return false;
} // threat method

 protected void outputBoard()
{
   System.out.println("Solution for board of size " + boardSize + ":");

   for (int row = 0; row < boardSize; row++)
   {
      for (int column = 0; column < boardSize; column++)
      {
         if (board[row][column])
         {
            System.out.print("Q ");
         }
         else
         {
            System.out.print(". ");
```

```
                }
             }

             System.out.println();
          }

          System.out.println();
       } // outputBoard method

        public void solve()
       {
          if (placeQueen(0))
          {
             outputBoard();
          }
          else
          {
             System.out.println("There is no solution possible");
          }
       } // solve method

        public static void main (String[] args)
       {
          Queens board;

          Scanner input = new Scanner (System.in);
          System.out.print("Enter the board Size: "); //Gain Size of board from user
          int boardSize = input.nextInt();
          input.close();

          board = new Queens(boardSize);
          board.solve();
       } // main method
    } /*Queens class*/
```

Which may result in an output like (assuming that an input of 8 was registered (i.e 8x8 board)):

code:

```
Enter the board Size: 8
Solution for board of size 8:
Q . . . . . . .
. . . . . . Q .
. . . . Q . . .
. . . . . . . Q
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .
. . Q . . . . .
```

In the main method we start with a vacant board by calling the constructor giving the board size as inputted by the user. We then call the solve() method to produce a solution for that board.

The method placeQueen is a recursive method for placing queens on the board from the column specified by column to the last column on the board given that queens have already been placed safely in column 0 to column - 1. The placeQueen method returns a boolean value that indicates whether it was possible to safely place queens from column to the end of the board. This method is recursive and reached its base case when the column is equal to the boardSize. The placeQueen method uses the boolean method *threat* which returns value true if a square is threatened by a queen in another column.

The method outputBoard provides the solutions and procedures a simpler representation of the board. This program searches for one solution. A variant (amongst many) would be to search for all solutions, which i will leave up to you (hint: there are 92 solutions).

Explore ...

DtY

Posted: Fri Jun 26, 2009 12:28 am Post subject: RE:Short Tutorial: Backtracking With Recursion

Pretty cool tutorial, I went through it, and wanted to see if I could rewrite it without looking at your code.
If anyone is interested, here it is in C:

c:

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>

//Checks if the queen can go there
//Looks at the row, colum, and diagonals
bool canGoHere(bool* board, int size, int row, int col) {
    int i;
    int irow, icol;

    for (irow=0; irow<size; irow++) {
        for (icol=0; icol<size; icol++) {
            int pos = icol*size+irow;
            if ( (irow == row) && (board[pos]) ) { return false; }
            if ( (icol == col) && (board[pos]) ) { return false; }
            if ( (abs(irow-row) == abs(icol-col)) && (board[pos]) ) { return false; }
        }
    }

    //Or else, we can go there!
    return true;
}

void showboard(bool* board, int size) {
    int row, col;
    for (row=0; row<size; row++) {
        for (col=0; col<size; col++) {
            if (board[col*size+row])
                printf("Q");
            else
                printf(".");
        }
        printf("\n");
    }
    puts("--");
}

bool placeQueen(int row, bool* board, int size) {
    showboard(board, size);
    int col = 0;
    if (row >= size) { return true; }
    //Try each column to see if the queen can go there
    for (col=0; col<size; col++) {
        if (col > 0) { board[(col-1)*size+row] = false; /*Make sure that the one above is
cleared*/ }
        if (canGoHere(board, size, row, col)) {
            board[col*size+row] = true; //Yes, let us go here
            return placeQueen(row+1, board, size);
        }
    }
    return false;
}

int main(void) {
    int size;
    int i;
    bool* board = NULL;

    puts("How big of a board?");
    scanf("%d", &size);
    printf("You entered: %d\n", size);
    /* Create the board */
    board = malloc(size*size*sizeof(bool));
    for (i=0; i<(size*size); i++) {
        board[i] = false;
    }
    placeQueen(0, board, size);


    free(board);
    return 0;
}
```

I love recursion so much. It looks like it does absolutely nothing, but it does this. (+karma)

Shah-Cuber

Posted: Fri Jun 26, 2009 9:45 am Post subject: Re: Short Tutorial: Backtracking With Recursion

@DtY: Very Nice, I liked the fact that you took this as a nice concept to demonstrate the skills learned through the somewhat understandable tutorial, and applied it yourself, and made an awesome alternative program
Overall, very well done, and i hope other people can also understand what i tried to teach out of boredom here >_>
Thanks for your contribution

Shah-Cuber

Posted: Fri Jun 26, 2009 9:52 am Post subject: Re: Short Tutorial: Backtracking With Recursion

Another tip, you may have realized that when placing n queens on an nxn chessboard, solutions exist only for n = 1 or n ≥ 4, so there is no solution for a 2x2 chess board, or a 3x3 chess board ...

Shah-Cuber

Posted: Fri Jun 26, 2009 3:56 pm Post subject: Re: Short Tutorial: Backtracking With Recursion

Alternative problems like this, amongst many, may include:

Bishops problem: Similar to Queens problem, except it's with bishop notation.
http://mathworld.wolfram.com/BishopsProblem.html

Kings problem: Determining how many nonattacking kings can be placed on an nxn chessboard.
http://mathworld.wolfram.com/KingsProblem.html

Knights problem: Determining how many nonattacking knights can be placed on an nxn chessboard.
http://mathworld.wolfram.com/KnightsProblem.html

Rooks problem: ... you get the picture ...
http://mathworld.wolfram.com/RooksProblem.html

You may even combine some of these concepts, to make a new problem, and see what happens ...

Kharybdis

Posted: Fri Jun 26, 2009 4:28 pm Post subject: RE:Short Tutorial: Backtracking With Recursion

hmm.. chess problems are fun.

Shah-Cuber

Posted: Fri Jun 26, 2009 4:51 pm Post subject: Re: Short Tutorial: Backtracking With Recursion

OHHH, solution for a 30x30 chessboard, hmmm, reminds me of flea circus from Project Euler =P

code:

```

Q . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . Q . . . . . . . . . . . . . . . . . . . . . . . . . .
. Q . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . Q . . . . . . . . . . . . . . . . . . . . . . . . .
. . Q . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
. . . . . . . . . . . . . . . . . . . . . . . Q . . . . . .
. . . . . . . . . Q . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . Q . . . . . . .
. . . . . Q . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . Q . . . . . . . .
. . . . . . Q . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . Q . . . . . . . . .
. . . . . . . Q . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . Q . .
. . . . . . . . Q . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . Q . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . Q . . .
. . . . . . . . . . . . . . . . . . . . . . . . Q . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . Q
. . . . . . . . . . . . . . . . . . . . . . . . . Q . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . Q .
. . . . . . . . . . . . . . Q . . . . . . . . . . . . . . .
. . . . . . . . . . Q . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . Q . . . . . . . . . . . . . .
. . . . . . . . . . . . . Q . . . . . . . . . . . . . . . .
. . . . . . . . . . . Q . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . Q . . . . . . . . . . . .
. . . . . . . . . . . . Q . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . Q . . . . . . . . . . .
. . . . . . . . . . . . . . . . Q . . . . . . . . . . . . .
```

Shah-Cuber

Posted: Sat Jun 27, 2009 7:35 pm Post subject: Re: Short Tutorial: Backtracking With Recursion

Ohhh, just found a nice problem on SPOJ, related to this ...
http://www.spoj.pl/problems/QKP/
I'm on it 😀

Analysis Mode

Posted: Sat Jun 27, 2009 10:37 pm Post subject: Re: Short Tutorial: Backtracking With Recursion

here's a more difficult one

http://www.spoj.pl/problems/QUEEN/

Shah-Cuber

Posted: Mon Jun 29, 2009 1:48 pm Post subject: Re: Short Tutorial: Backtracking With Recursion

Oh nice find Analysis Mode, i'll start on that too 😀 thx

andersbranderud

Posted: Tue Sep 08, 2009 9:06 am Post subject: RE:Short Tutorial: Backtracking With Recursion

Find your tutorial via google.
Thanks for the tutorial!

Anders Branderud
bloganders.blogspot.com

Shah-Cuber

Posted: Tue Sep 08, 2009 12:25 pm Post subject: Re: Short Tutorial: Backtracking With Recursion

No problem 😃

dumisan

Posted: Sun Jan 09, 2011 8:11 am Post subject: Re: Short Tutorial: Backtracking With Recursion

I'd like to add here a version of nqueen problem solution useing backtracking that gives all the posible solutions, each solution as a permutation but is not useing recursion(kinda not likeing it)
so something like (1 5 8 6 3 7 2 4) means that on the first row the queens is placed in the first position, on the second row the queen is on th 5th position so on ..

CODE:

```
public class nquens {
 public static void main(String args[]) {
   System.out.println("give n");
    java.util.Scanner in = new java.util.Scanner(System.in);
    int n = in.nextInt();
    in.close();
    if((n==2)||(n==3)) System.out.println("The problem have no solutions");
    else {
    int sp = 1;
    int[] x = new int[n];
    int k = 0;
    x[0] = 0;
    while (k >= 0) {
        x[k] = x[k] + 1;
        if (x[k] > n) {
            k = k - 1;
        } else {
            sp = 1;
            for (int i = 0; i <= k-1; i++)
                if ((x[i] == x[k])||(Math.abs(x[k]-x[i])==Math.abs(k-i)))
                    sp = 0;
            if (sp == 1) {
                if (k == n-1 ) {
                    for (int i = 0; i < n; i++) {
                        System.out.print(x[i]);
                    } //end of the for
                    System.out.println();
                } // if (k==n-1)
                else {
                    k = k + 1;
                    x[k] = 0;
                    } // else from if(k==n)
            } //if(sp==1)
        } // else from if(x[k]==n)
    } //end of the while
    }//end of if n==2 || n==3
} //end of main
}
```

well is not the nicest java implementation 😟 but works well 😉

jerseywhore

Posted: Tue Apr 26, 2011 1:15 am Post subject: Re: Short Tutorial: Backtracking With Recursion

Hi, thanks for this tutorial! Just one question: are you using a multi-dimensional array? I'm studying Java right now, and our professor skipped the chapter about multi-dimensional arrays, so I'm not sure how it looks like.

Insectoid

Posted: Tue Apr 26, 2011 10:16 am Post subject: RE:Short Tutorial: Backtracking With Recursion

code:

```
  protected boolean[][] board;
```

Yeah, this is a 2-D array. You'll use them a lot.

Display posts from previous: All Posts Oldest First Go

Index -> General Programming

Page 1 of 2 [16 Posts]

Goto page 1, 2 Next

Jump to: - General Programming

Style: subGrey Go

Search: Go